An Algorithm for Multipath Computation using Distance-Vectors with Predecessor Information

SRINIVAS VUTUKURY
vutukury@cse.ucsc.edu
Computer Sciences Department
University of California
Santa Cruz, CA 95064

J.J. GARCIA-LUNA-ACEVES
jj@cse.ucsc.edu
Computer Engineering Department
University of California
Santa Cruz, California 95064
Networking and Security Center
Sun Microsystems Laboratories
Palo Alto, California 94303

Abstract—**Routing algorithms in the IP Internet provide a single path between each source-destination pair and where more than one path is provided, they are paths of equal length. Single-path routing is inherently slow in responding to congestion and temporary traffic bursts; multiple paths are better suited to handle congestion. Also the paths provided in RIP and OSPF are not free of loops during times of network transition, which can be debilitating to network performance. We present a distributed routing algorithm for computing multiple paths that need not have equal length between each source-destination pair in a computer network such that they are loop-free at every instant—in steady state as well as during network transitions. The algorithm is scalable to large networks as it uses only one-hop synchronization which is unlike diffusing computations that require internodal synchronization spanning multiple hops. The safety and liveness properties of the algorithm are proven and its complexity is analyzed.**

I. INTRODUCTION

The most popular routing protocols used in today's internets are based on the exchange of vectors of distances (e.g., RIP [7] and EIGRP [2]) or topology maps (e.g., OSPF [11]). RIP and many other routing protocols based on the distributed Bellman-Ford algorithm (DBF) for shortest-path computation suffer from the *bouncing effect* and the *counting-to-infinity* problems, which limits their applicability to small networks using hop count as the measure of distance. OSPF and algorithms based on topology-broadcast (e.g., [15], [12]) incur too much communication overhead, which forces the network administrators to partition the network into areas connected by a backbone. This makes OSPF complex in terms of router configuration required. EIGRP uses a loop-free routing algorithm called DUAL [3], which is based on internodal coordination that can span multiple hops.

In addition to DUAL, several algorithms based on distance vectors have been proposed to overcome the counting-to-infinity problem of DBF [14], [10], [9], [17]. All of these algorithms rely on exchanging queries and replies along multiple hops, a technique that is sometimes called *diffusing computations*, because it has its origin in Dijkstra and Scholten's basic algorithm [1].

A couple of routing algorithms have been proposed that operate using partial topology information [4], [6] to eliminate the main limitation of topology-broadcast algorithms. Furthermore, several distributed shortest-path algorithms [8], [13], [5] have been proposed that use the distance and second-to-last hop to destinations as the routing information exchanged among nodes. These algorithms are often called path-finding algorithms or source-tracing algorithms. All these algorithms eliminate DBF's counting to infinity problem, and some of them [5] are more efficient that any of the routing algorithms based on link-state information proposed to date. Furthermore, LPA [5] is loop-free at every instant.

This work was supported in part by the Defense Advanced Research Projects Agency (DARPA) under grants F30602-97-1-0291 and F19628-96-C-0038.

With the exception of DASM [17], all of the above routing algorithms focus on the provision of a single path to each destination. A drawback of DASM, however, is that it uses multi-hop synchronization, which limits its scalability. Recently, we presented MPDA [16] which is the first routing algorithm based on link-states that provides multiple loop-free paths using one-hop synchronization. In this paper, we present a variant of MPDA called MPATH, which is the first routing algorithm based on distance vectors that (a) provides multiple paths of unequal cost to each destination that are free of loops at every instant — in steady state as well as during network transitions, and (b) uses a synchronization mechanism that spans only one hop, which makes it more scalable than routing algorithms based on diffusing computations spanning multiple hops. MPATH is a path-finding algorithm, and differs from prior similar algorithms in the invariants used to ensure multiple loop-free paths of unequal cost. The peculiar differences between MPATH and MPDA is a result of the differences in the kind of information that nodes exchange.

Section II describes MPATH. Section III presents the correctness proofs showing that MPATH is loop-free at every instant, safe, and live. Section IV analyzes the complexity of MPATH. Section V provides concluding remarks.

II. DISTRIBUTED MULTIPATH ROUTING ALGORITHM

A. Problem Formulation

A computer network is represented as a graph $G = (N, L)$ where N is set of nodes (routers) and L is the set of edges (links) connecting the nodes. A cost is associated with each link and can change over time, but is always positive. Two nodes connected by a link are called adjacent nodes or neighbors. The set of all neighbors of a given node i is denoted by N^i. Adjacent nodes communicate with each other using messages and messages transmitted over an operational link are received with no errors, in the proper sequence, and within a finite time. Furthermore, such messages are processed by the receiving node one at a time in the order received. A node detects the failure, recovery and link cost changes of each adjacent link within a finite time.

The goal of our distributed routing algorithm is to determine at each node i the successor set of i for destination j, which we denote by $S_j^i(t) \subseteq N^i$, such that the routing graph $SG_j(t)$ consisting of link set $\{(m,n)|n \in S_j^m(t),\ m \in N\}$ is free of loops at every instant t, even when link costs are changing with time. The routing graph $SG_j(t)$ for single-path routing is a sink-tree rooted at j, because the successor sets $S_j^i(t)$ have at most one member. In multipath routing, there can be more than one member in $S_j^i(t)$; therefore, $SG_j(t)$ is a directed acyclic graph with j as the sink node. There are potentially several $SG_j(t)$ for each destination j; however, the routing graph we are interested is defined by the successor sets $S_j^i(t) = \{k|D_j^k(t) < D_j^i(t), k \in N^i\}$,

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
1999		**00-00-1999 to 00-00-1999**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
An Algorithm for Multipath Computation using Distance-Vectors with Predecessor Information	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of California at Santa Cruz,Department of Computer Engineering,Santa Cruz,CA,95064**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**6**	

Procedure *INIT-PATH*

{*Invoked when the node comes up.*}

1. Initialize all tables.
2. Run $PATH$ algorithm.

End *INIT-PATH*

Algorithm *PATH*

{*Invoked when a message M is received from neighbor k, or an adjacent link to k has changed or when a node is initialized.*}

1. Run NTU to update neighbor tables.
2. Run MTU to update main tables.
3. For each destination j marked as *changed*,
   Add update entry [j, D_j^i, p_j^i] to the new message M'.
4. Within finite amount of time, send message M' to each neighbor.

End *PATH*

Fig. 1. The PATH Algorithm

where D_j^i is the shortest distance of node i to destination j. We call such a routing graph the *shortest multipath* for destination j.

After a series of link cost changes which leave the network topology in arbitrary configuration, the distributed routing algorithm should work to modify SG_j in such a way that it eventually converges to the shortest multipath of the new configuration, without ever creating a loop in SG_j during the process.

Because D_j^k is node k's local variable, its value has to be explicitly or implicitly communicated to i. If D_{jk}^i is the value of D_j^k as known to node i, the problem now becomes one of computing $S_j^i(t) = \{k | D_{jk}^i(t) < D_j^i(t)\}$. However, because of non-zero propagation delays, during network transitions there can be discrepancies in the value of D_j^k and its copy D_{jk}^i at i, which may cause loops to form in SG_j. To prevent loops, therefore, additional constraints must be imposed when computing S_j^i. We show later that if the successor set at each node i for each destination j satisfy certain conditions called loop-free invariant conditions, then the snapshot at time t of the routing graph $SG_j(t)$ implied by $S_j^i(t)$ is free of loops. Our solution to this problem consists of two parts: (1) computing D_j^i using a shortest-path routing algorithm called PATH and (2) extending it to compute S_j^i such that they satisfying loop-free invariant conditions at every instant.

B. Node Tables and Message Structures

As in DBF, nodes executing MPATH exchange messages containing distances to destinations. In addition to the distance to a destination, nodes also exchange the identity of the second-to-last node, also called predecessor node, which is the node just before the destination node on the shortest path. In this respect MPATH is akin to several prior algorithms [5], [13], [8], but differs in its specification, verification and analysis and, more importantly, in the multipath operation described in the next section.

The following information is maintained at each node i:

1. The *Main Distance Table* contains D_j^i and p_j^i, where D_j^i is the distance of node i to destination j and p_j^i is the predecessor to destination j on the shortest path from i to j. The table also stores for each destination j, the successor set S_j^i, feasible distance FD_j^i, reported distance RD_j^i and two flags *changed* and *report-it*.
2. The *Main Link Table* T^i is the node's view of the network and contains links represented by $(m,\ n,\ d)$ where $(m,\ n)$ is a link with cost d.
3. The *Neighbor Distance Table* for neighbor k contains D_{jk}^i and p_{jk}^i where D_{jk}^i is the distance of neighbor k to j as communicated by k and p_{jk}^i is the predecessor to j on the shortest path from k to j as notified by k.
4. The *Neighbor Link Table* T_k^i is the neighbor k's view of the network as known to i and contains link information derived from the distance and predecessor information in the neighbor distance table.
5. *Adjacent Link Table* stores the cost l_k^i of adjacent link to each neighbor k. If a link is down its cost is infinity.

Nodes exchange information using update messages which have the following format.

1. An update message can one or more update entries. An update entry is a triplet [j, d, p], where d is the distance of the node sending the message to destination j and p is the predecessor on the path to j.
2. Each message carries two flags used for synchronization: *query* and *reply*.

Procedure NTU

{*Called by PATH to process an event.*}

1. If event is a message M from neighbor k,
   a. For each entry [j, d, p] in M //Note $d = D_j^k$, $p = p_j^k$.)
      Set $D_{jk}^i \leftarrow d$ and $p_{jk}^i \leftarrow p$.
   b. For each destination j with an entry in M,
      Remove existing links (n, j) in T_k^i and add new link (m, j, d) to T_k^i, where $d = D_{jk}^i - D_{mk}^i$ and $m = p_{jk}^i$.
2. If the event is an adjacent link-status change, update l_k^i and clear neighbor tables of k, if link is down.

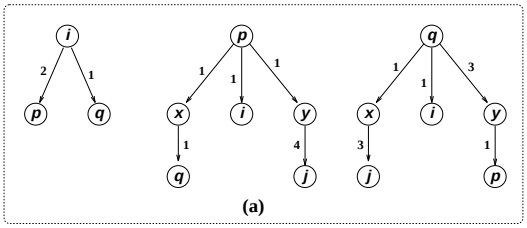
End NTU

Fig. 2. Neighbor Table Update Algorithm

C. Computing D_j^i

As mentioned earlier, our strategy is to first design a shortest-path routing algorithm and then make the multipath extensions to it. This subsection describes our shortest-path algorithm PATH and the next subsection describes the multipath extensions. Figure 1 shows the pseudocode of PATH. INIT-PATH is called at node startup to initialize the tables; distances are initialized to infinity and node identities to a null value. PATH is executed in response to an event that can be either a receipt of an update message from a neighbor or detection of an adjacent link cost or link status (up/down) change. PATH invokes procedure NTU, described in Figure 2, which first updates the neighbor distance tables and then updates T_k^i with links $(m,\ n,\ d)$ where $d = D_{nk}^i - D_{mk}^i$ and $m = p_{nk}^i$. PATH then invokes procedure MTU, specified in Figure 5, which constructs T^i by merging the topologies T_k^i and the adjacent links l_k^i.

The merging process is straightforward if all neighbor topologies T_k^i contain consistent link information, but when two or more neighbors link tables contain conflicting information regarding a particular link, the conflict must be resolved. Two neighbor tables are said to contain conflicting information regarding a link, if either both report the link with different cost or one reports the link and the other does not. Conflicts are resolved as follows: if two or more neighbor link tables contain conflicting information of link $(m,\ n)$, then T^i is updated with link information reported by the neighbor k that offers the shortest distance from the node i to the head node m of the link, i.e., $l_k^i + D_{mk}^i = min\{l_k^i + D_{mk}^i | k \in N^i\}$. Ties are broken in a *consistent* manner; one way is to break ties always in favor of lower address neighbor. Because i itself is the head of the link for adjacent links, any information about an adjacent link supplied by neighbors will be overridden by the most current information about the link available to node i. Figure 4 shows the significance of the tie-breaking rule.



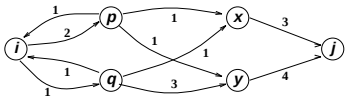


Table showing the preferred neighbors.

Destination	p	q	x	y	j
Distance	2	1	2	3	5
Pref. Nbr	p	q	q	p	q

Fig. 3. Example illustrating the main table update procedure. (a) Shows the adjacent links and neighbor tables of node i. (b) Shows the main link table i after merging the neighbor tables

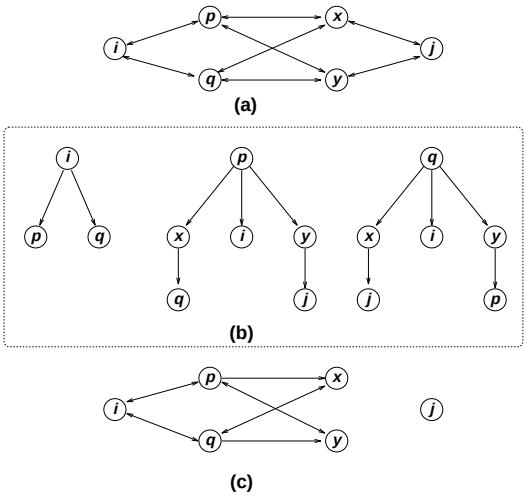


Fig. 4. Significance of the tie-breaking Rule. (a) An example network with unit link costs. (b) Node i has the costs of its adjacent links and the shortest path trees of its neighbors p and q. The distances of nodes x and y from i is identical through both neighbors p and q. (c) If MTU breaks ties in arbitrary manner while constructing T^i, it may choose p as the preferred neighbor for node x and choose q as preferred neighbor for node y, resulting in a graph that has no path from i to j. Ties, therefore, cannot be broken in arbitrary manner.

After merging the topologies, MTU runs Dijkstra's shortest path algorithm to find the shortest path tree and deletes all links from T^i that are not in the tree. Because there can be more than one shortest-path tree, while running Dijkstra's algorithm ties are again broken in a consistent manner. The distances D^i_j and predecessors p^i_j can then be obtained from T^i. The tree is compared with the previous shortest path tree and only the differences are then reported to the neighbors. If there are no differences, no updates are reported. Eventually all tables converge such that D^i_j give the shortest distances and all message activity will cease. The proofs are given in section III.

D. Computing S^i_j

In this subsection, the final desired routing algorithm MPATH is derived by making extensions to PATH. MPATH computes the successor sets S^i_j by enforcing the Loop-free Invariant conditions described below and using a neighbor-to-neighbor synchronization.

Procedure MTU

1. Clear link table T^i.
2. For each node $j \neq i$ occurring in at least one T^i_k,
   a. Find $MIN \leftarrow min\{D^i_{jk} + l^i_k | k \in N^i\}$.
   b. Let n be such that $MIN = (D^i_{jn} + l^i_n)$. Ties are broken *consistently*. Neighbor n is the preferred neighbor for destination j. For each link (j, v, d) in T^i_n, Add link (j, v, d) to T^i.
3. Update T^i with each link l^i_k.
4. Run Dijkstra's shortest path algorithm on T^i to find new D^i_j, and p^i_j.
5. For each destination j, if D^i_j or p^i_j changed from previous value, set *changed* and *report-it* flags for j.

End MTU

Fig. 5. Main Table Update Algorithm

Let FD^i_j, called the feasible distance, be an 'estimate' of the distance of node i to node j in the sense that FD^i_j is equal to D^i_j when the network is in stable state, but to prevent loops during periods of network transitions, it is allowed to be temporarily differ from D^i_j.

Loop-free Invariant Conditions(LFI)[16]:

$$FD^i_j(t) \quad \leq \quad D^k_{ji}(t) \qquad k \in N^i \tag{1}$$

$$S^i_j(t) \quad = \quad \{\, k \mid D^i_{jk}(t) < FD^i_j(t) \,\} \tag{2}$$

The invariant conditions (1) and (2) state that, for each destination j, a node i can choose a successor whose distance to j, as known to i, is less than the distance of node i to j that is known to its neighbors.

Theorem *1:* [16] If the LFI conditions are satisfied at any time t, the $SG_j(t)$ implied by the successor sets $S^i_j(t)$ is loop-free.

Proof: Let $k \in S^i_j(t)$ then from (2) we have

$$D^i_{jk}(t) \quad < \quad FD^i_j(t) \tag{3}$$

At node k, because node i is a neighbor, from (1) we have

$$FD^k_j(t) \quad \leq \quad D^i_{jk}(t) \tag{4}$$

Combining (3) and (4) we get

$$FD^k_j(t) \quad < \quad FD^i_j(t) \tag{5}$$

Eq.(5) states that, if k is a successor of node i in a path to destination j, then k's feasible distance to j is strictly less than the feasible distance of node i to j. Now, if the successor sets define a loop at time t with respect to j, then for some node p on the loop, we arrive at the absurd relation $FD^p_j(t) < FD^p_j(t)$. Therefore the LFI conditions are sufficient for loop-freedom. ■

The invariants used in LFI are independent of whether the algorithm uses link states or distance vectors; in link-state algorithms, such as MPDA, the D^i_{jk} are computed locally from the link-states communicated by the neighbors while in distance-vector algorithms, like the MPATH presented here, the D^i_{jk} are directly communicated.

The invariants (1) and (2) suggest a technique for computing $S^i_j(t)$ such that the successor graph $SG_j(t)$ for destination j is loop-free at every instant. The key is determining $FD^i_j(t)$ in Eq. (1), which requires node i to know $D^k_{ji}(t)$, the distance from i to node j in the topology

Procedure *INIT-MPATH*
{*Invoked when the node comes up.*}
1. Initialize tables and run MPATH.
End *INIT-MPATH*

Algorithm $MPATH$
{*Invoked when a message M is received from neighbor* k*, or an adjacent link to* k *has changed.*}
1. Run NTU to update neighbor tables.
2. Run MTU to obtain new D_j^i and p_j^i.
3. If node is *PASSIVE* or node is *ACTIVE* $\wedge$ last reply arrived,
   Reset *goactive* flag.
   For each destination j marked as *report-it*,
   a. $FD_j^i \leftarrow min\{D_j^i, RD_j^i\}$
   b. If $D_j^i > RD_j^i$, Set *goactive* flag.
   c. $RD_j^i \leftarrow D_j^i$
   d. Add $[j, RD_j^i, p_j^i]$ to message M'.
   e. Clear *report-it* flag for j.
   Otherwise, the node is ACTIVE and waiting for more replies,
   For each destination j marked as *changed*,
   f. $FD_j^i \leftarrow min\{D_j^i, FD_j^i\}$
4. For each destination j marked as *changed*,
   a. Clear *changed* flag for j
   b. $S_j^i \leftarrow \{k | D_{jk}^i < FD_j^i\}$
5. For each neighbor k,
   a. $M'' \leftarrow M'$.
   b. If event is a *query* from k, Set *reply* flag in M''.
   c. If *goactive* set, Set *query* flag in M''.
   d. If M'' non-empty, send M'' to k.
6. If *goactive* set, become $ACTIVE$, otherwise become $PASSIVE$.
End $MPATH$

Fig. 6. Multi-path Loop-free Routing Algorithm

table T_i^k that node i communicated to neighbor k. Because of non-zero propagation delay, T_i^k is a time-delayed version of T^i. We observe that, if node i delays updating of FD_j^i with D_j^i until k incorporates the distance D_j^i in its tables, then FD_j^i satisfies the LFI condition.

Pseudocode for MPATH is shown in Figure 6. MPATH enforces the LFI conditions by synchronizing the exchange of update messages among neighbors using *query* and *reply* flags. If a node sends a message with a *query* bit set, then the node must wait until a *reply* is received from all its neighbors before the node is allowed to send the next update message. The node is said to be in ACTIVE state during this period. The inter-neighbor synchronization used in MPATH spans only one hop, unlike algorithms that use diffusing computation that potentially span the whole network(e.g., DASM [17]).

Assume that all nodes are in PASSIVE state initially with correct distances to all other nodes and that no messages are in transit or pending to be processed. The behavior of the network where every node runs MPATH is such that when a finite sequence of link cost changes occurs in the network within a finite time interval, some or all nodes to go through a series of PASSIVE-to-ACTIVE and ACTIVE-to-PASSIVE state transitions, until eventually all nodes become PASSIVE with correct distances to all destinations.

Let a node in PASSIVE state receive an event resulting in changes in its distances to some destinations. Before the node sends an update message to report new distances, it checks if the distance D_j^i to any destination j has increased above the previously reported distance RD_j^i. If none of the distances increased, then the node remains in PASSIVE state. Otherwise, the node sets the *query* flag in the update message, sends it, and goes into ACTIVE state. When in ACTIVE state, a node cannot send any update messages or *add* neighbors to any successor set. After receiving replies from all its neighbors the node is allowed to modify the successor sets and report any changes that may have occurred since the time it has transitioned to ACTIVE state, and if none of the distances increased beyond the reported distance, the node transitions to PASSIVE state. Otherwise, the node sends the next update message with the *query* bit set and becomes ACTIVE again, and the whole cycle repeats. If a node receives a message with the *query* bit set when in PASSIVE state, it modifies its tables and then sends back an update message with the *reply* flag set. Otherwise, if the node happens to be in ACTIVE state, it modifies the tables but because the node is not allowed to send updates when in ACTIVE state, the node sends back an empty message with no updates but the *reply* bit set. If a reply from a neighbor is pending when the link to the neighbor fails then an implicit reply is assumed, and such a reply is assumed to report an infinite distance to the destination. Because replies are given immediately to queries and replies are assumed to be given upon link failure, deadlocks due to inter-neighbor synchronization cannot occur. Eventually, all nodes become PASSIVE with correct distances to destinations, which we prove in the next section.

III. Correctness of MPATH

The following properties of MPATH must be proved: (1) MPATH eventually converges with D_j^i giving the shortest distances and (2) the successor graph SG_j is loop-free at every instant and eventually converges to the shortest multipath. PATH works essentially like PDA[16] except that the kind of update information exchanged is different; PDA exchanges link-state while PATH exchanges distance-vectors with predecessor information. The correctness proof of PATH is identical to PDA and are reproduced here for correctness. The convergence of MPATH directly follows from the convergence of PATH because extensions to MPATH are such that update messages in MPATH are only delayed a finite amount of time.

Definitions: The n-hop minimum distance of node i to node j in a network is the minimum distance possible using a path of n hops(links) or less. A path that offers the n-hop minimum distance is called n-hop minimum path. If there is no path with n hops or less from node i to j then the n-hop minimum distance from i to j is undefined. An n-hop minimum tree of a node i is a tree in which node i is the root and all paths of n hops or less from the root to any other node is an n-hop minimum path.

Let $\mathbf{G}$ denote the final topology of the network, as seen by an omniscient observer, after all link changes occurred. (We use bold font to refer to quantities in $\mathbf{G}$). Without loss of generality, assume $\mathbf{G}$ is connected; if $\mathbf{G}$ is disconnected, the proof applies to each connected component independently.

We say that a router i *knows at least* the n-hop minimum tree, if the tree contained in its main link table T^i is at least an n-hop minimum tree rooted at i in $\mathbf{G}$ and there are at least n nodes in T^i that are reachable from the root i. Note that T^i is such that the links with head nodes that are more than n hops away from i may have costs that do not agree with the link costs in $\mathbf{G}$.

Theorem *2:* If node i has adjacent link costs that agree with $\mathbf{G}$ and for each neighbor k, T_k^i represents at least an $(n-1)$-hop minimum tree, then after the execution of MTU, the minimum cost tree contained in T^i is at least an n-hop minimum tree.

Proof: The proof is identical to the proof of Lemma 1 in [16] and is provided in the appendix for convenient reference. ■

Theorem *3:* A finite time after the last link cost change in the network, the main topology T^i at each node i gives the correct shortest paths to all known destinations.

Proof: The proof is identical to the proof of Theorem 2 in [16]

and is provided in the appendix for convenient reference. ■

A node generates update messages only to report changes in distances and predecessor, so after convergence no messages will be generated. The following theorems show that MPATH provides instantaneous loop-freedom and correctly computes the shortest multipath.

Theorem *4:* For the algorithm MPATH executed at node i, let t_n be the time when RD_j^i is updated and reported for the n-th time. Then, the following conditions always hold.

$$FD_j^i(t_n) \leq min\{RD_j^i(t_{n-1}), RD_j^i(t_n)\} \quad (6)$$

$$FD_j^i(t) \leq FD_j^i(t_n) \qquad t \in [t_n, t_{n+1}) \quad (7)$$

Proof: From the working of MPATH in Fig. 6, we observe that RD_j^i is updated at line 3c when (a) the node goes from PASSIVE-to-ACTIVE because of one or more distance increases (b) the node receives the last reply and goes from ACTIVE-to-PASSIVE state (c) the node is in PASSIVE state and remains in PASSIVE state because the distance did not increase for any destination (d) the node receives the last reply but immediately goes into ACTIVE state. The reported distance RD_j^i remains unchanged during the ACTIVE phase. Because FD_j^i is updated at line 3a each time RD_j^i is updated at line 3c, Eq. (6) follows. When the node is in ACTIVE phase, FD_j^i may also be modified by the statement on line 3f, which implies Eq. (7). ■

Theorem *5:* (Safety property) At any time t, the successor sets $S_j^i(t)$ computed by MPATH are loop-free.

Proof: The proof is based on showing that the FD_j^i and S_j^i computed by MPATH satisfy the LFI conditions. Let t_n be the time when RD_j^i is updated and reported for the n-th time. The proof is by induction on the interval $[t_n, t_{n+1}]$. Let the LFI condition be true up to time t_n, we show that

$$FD_j^i(t) \leq D_{ji}^k(t) \qquad t \in [t_n, t_{n+1}] \quad (8)$$

From Theorem 4 we have

$$FD_j^i(t_n) \leq min\{RD_j^i(t_{n-1}), RD_j^i(t_n)\} \quad (9)$$

$$FD_j^i(t_{n+1}) \leq min\{RD_j^i(t_n), RD_j^i(t_{n+1})\} \quad (10)$$

$$FD_j^i(t) \leq FD_j^i(t_n) \qquad t \in [t_n, t_{n+1}) \quad (11)$$

Combining the above equations we get

$$FD_j^i(t) \leq min\{RD_j^i(t_{n-1}), RD_j^i(t_n)\} \quad t \in [t_n, t_{n+1}] \quad (12)$$

Let t' be the time when message sent by i at t_n is received and processed by neighbor k. Because of the non-zero propagation delay across any link, t' is such that $t_n < t' < t_{n+1}$ and because RD_j^i is modified at t_n and remains unchanged in (t_n, t_{n+1}) we get

$$RD_j^i(t_{n-1}) \leq D_{ji}^k(t) \qquad t \in [t_n, t') \quad (13)$$

$$RD_j^i(t_n) \leq D_{ji}^k(t) \qquad t \in [t', t_{n+1}] \quad (14)$$

From Eq. (13) and (14) we get

$$min\{RD_j^i(t_{n-1}), RD_j^i(t_n)\} \leq D_{ji}^k(t) \quad t \in [t_n, t_{n+1}] \quad (15)$$

From (12) and (15) the inductive step (8) follows. Because $FD_j^i(t_0) \leq D_{ji}^k(t_0)$ at initialization, from induction we have that $FD_j^i(t) \leq D_{ji}^k(t)$ for all t. Given that the successor sets are computed based on FD_j^i, it follows that the LFI conditions are always satisfied. According to the Theorem 1 this implies that the successor graph SG_j is always loop-free. ■

Theorem *6:* (Liveness property) A finite time after the last change in the network, the D_j^i give the correct shortest distances and $S_j^i = \{k | D_j^k < D_j^i, k \in N^i\}$.

Proof: The proof is similar to the proof of Theorem 4 in [16] and is provided in the appendix for convenience. ■

IV. Complexity Analysis

The main difference between PATH and MPATH is that the update messages sent in MPATH are delayed a finite amount of time in order to enforce the invariants. As a result, the complexity of PATH and MPATH are essentially the same and are therefore collectively analyzed.

The *storage complexity* is the amount of table space needed at a node. Each one of the N^i neighbor tables and the main distance table has size of the order $O(|N|)$ and the main link table T^i can grow, during execution of MTU, to size at most $|N^i|$ times $O(|N|)$. The storage complexity is therefore of the order $O(|N^i||N|)$.

The *time complexity* is the time it takes for the network to converge after the last link cost change in the network. To determine time complexity we assume the computation time to be negligible as compared to the communication times. If t_n is the time when every node has the n-hop minimum tree, because every node processes and reports changes in finite time $|t_{n+1} - t_n|$ is bounded. Let $|t_{n+1} - t_n| \leq \theta$ for some finite constant θ. From theorem 3, the convergence time can be at most $|N|\theta$ and, hence, the time complexity is $O(|N|)$.

The *computation complexity* is the time taken to build the node's shortest path tree in T^i from the neighbor tables T_k^i. Updating of T^i with T_k^i information is $O(|N^i||N|)$ operation and running Dijkstra on T^i takes $O(|N^i||N|log(|N|))$. Therefore the computation complexity is $O(|N^i||N| + |N^i||N|log(|N|))$.

The *communication complexity* is the number of update messages required for propagating a set of link-cost changes. The analysis for multiple link-cost changes is complex because of the sensitivity to the timing of the changes. So, we provide the analysis only for the case of single link-cost change. A node removes a link from its shortest path tree if only a shorter path using two or more links is discovered and once discovered the path is remembered. Therefore, a removed link will not be added again to the shortest path which means that a link can be included and deleted from the shortest path by a node at most one time. Because nodes report each change only once to each neighbor, an update message can travel only once on a link and therefore the number of messages sent by a node can be at most $O(|E|)$. For certain topologies and sensitively timed sequence of link cost changes the amount of communication required by PATH can be exponential. Humblet [8] provides an example that exhibits such behavior, and though PATH is different from the shortest-path algorithm presented in that paper, we note that PATH is not immune from such exponential behavior. However, we believe such scenarios require sensitively timed link-cost changes which are very unlikely to occur in practice. If necessary, a small hold-down time before sending update messages may be used to prevent such behavior.

V. Concluding Remarks

We have presented the first routing algorithm based on distance information that provides multiple paths that need not have equal costs and that are loop-free at every instant, without requiring inter-nodal synchronization spanning more than one hop. The loop-free invariant conditions presented here are quite general and can be used with existing internet protocols. The multiple successors that MPATH makes available at each node can be used for traffic load-balancing, which as we have shown using other algorithms (MPDA [16]) is necessary for minimizing delays in a network. MPATH can therefore be used as an alternative to MPDA to get similar performance. In a future work we intend to compare the performance of the three multipath routing algorithms MPATH, MPDA and DASM[17] in terms of control message overhead and convergence times and analyze their relative merits.

References

[1] E.W.Dijkstra and C.S.Scholten. Termination Detection for Diffusing Computations. *Information Processing Letters*, 11:1–4, August 1980.

[2] D. Farinachi. Introduction to enhanced IGRP(EIGRP). *Cisco Systems Inc.*, July 1993.

[3] J.J. Garcia-Luna-Aceves. Loop-Free Routing Using Diffusing Computations. *IEEE/ACM Trans. Networking*, 1:130–141, February 1993.

[4] J.J. Garcia-Luna-Aceves and J. Behrens. Distributed, scalable routing based on vectors of link states. *IEEE Journal on Selected Areas in Communications*, October 1995.

[5] J.J. Garcia-Luna-Aceves and S. Murthy. A path-finding algorithm for loop-free routing. *IEEE/ACM Trans. Networking*, February 1997.

[6] J.J. Garica-Luna-Aceves and M. Spohn. Scalable link-state internet routing. *Proc. International Conference on Network Protocols*, October 1998.

[7] C. Hendrick. Routing Information Protocol. *RFC*, 1058, june 1988.

[8] P. A. Humblet. Another Adaptive Distributed Shortest Path Algorithm. *IEEE Trans. Commun.*, 39:995–1003, June 91.

[9] J. M. Jaffe and F. H. Moss. A Responsive Distributed Routing Algorithm for Computer Networks. *IEEE Trans. Commun.*, 30:1758–1762, July 1982.

[10] P. M. Merlin and A. Segall. A Failsafe Distributed Routing Protocol. *IEEE Trans. Commun.*, 27:1280–1287, September 1979.

[11] J. Moy. OSPF Version 2. *RFC*, 1247, August 1991.

[12] R. Perlman. Fault-tolerant broadcast of routing information. *Computer Networks and ISDN*, 7, 1983.

[13] B. Rajagopalan and M. Faiman. A Responsive Distributed Shortest-Path Routing Algorithm with Autonomous Systems. *Internetworking: Research and Experience*, 2:51–69, March 1991.

[14] A. Segall. Optimal distributed routing for virtual line-switched data networks. *IEEE Trans. Commun.*, 27:201–209, January 1979.

[15] J. Spinelli and R. Gallager. Event Driven Topology Broadcast without Sequence Numbers. *IEEE Trans. Commun.*, 37:468–474, 1989.

[16] S. Vutukury and J.J. Garcia-Luna-Aceves. A Simple Approximation to Minimum Delay Routing. *Proc. of ACM SIGCOMM*, 1999.

[17] W. T. Zaumen and J.J. Garcia-Luna-Aceves. Loop-Free Multipath Routing Using Generalized Diffusing Computations. *Proc. IEEE INFOCOM*, March 1998.

APPENDIX

Proof of Theorem 2:

Let $\mathbf{H_n^i}$ denote an n-hop minimum tree rooted at node i in $\mathbf{G}$ and let $\mathbf{M_n^i}$ be the set of nodes that are within n hops from i in $\mathbf{H_n^i}$. Let $\mathbf{D_n^{i,j}}$ denote the distance of i to j in $\mathbf{H_n^i}$. Let $\mathbf{d_{ij}}$ be the cost of the link $i \rightarrow j$. Node i is called the head of the link $i \rightarrow j$. The notation $i \rightsquigarrow j$ indicates a path from i to j of zero or more links; if the path has zero links, then $i = j$. The length of path $i \rightsquigarrow j$ is the sum of costs of all links in the path.

Property *1:* From the principle of optimality (the sub-path of a shortest path between two nodes is also the shortest path between the end nodes of the sub-path), if H and H' are two n-hop minimum trees rooted at node i and M and M' are sets of nodes that are within n hops from i in H and H' respectively, then $M = M' = \mathbf{M_n^i}$ and $\mathbf{M_n^i} \geq \mathbf{n}$. For each $j \in \mathbf{M_n^i}$ the length of path $i \rightsquigarrow j$ in both H and H' is equal to $\mathbf{D_n^{i,j}}$. For $h \geq n$, $\mathbf{D_h^{i,j}} \leq \mathbf{D_n^{i,j}}$.

Let $A^i = \bigcup_{k \in N^i} A_k^i$, where A_k^i is the set of nodes in T_k^i. Because T_k^i is at least an $(n-1)$-hop minimum tree and node i can appear at most once in each of A_k^i, each A_k^i has at least $n-1$ unique elements. Therefore, A^i has at least $n-1$ elements.

Let M_n^i be the set of $n-1$ nearest elements to node i in A^i. That is, $M_n^i \subseteq A^i$, $|M_n^i| = n-1$, and for each $j \in M_n^i$ and $v \in A^i - M_n^i$, $min\{D_{jk}^i + l_k^i | k \in N^i\} \leq min\{D_{vk}^i + l_k^i | k \in N^i\}$.

To prove the theorem it is sufficient to prove the following:

1. Let G_n^i represent the graph constructed by MTU on lines 2 and 3. (i.e., before applying Dijkstra in line 4). For each $j \in M_n^i$ there is a path $i \rightsquigarrow j$ in G_n^i such that its length is at most $\mathbf{D_n^{i,j}}$.
2. After running Dijkstra on G_n^i on line 4 in MTU, the resulting tree is at least an n-hop minimum tree.

Let us first assume part 1 is true and prove part 2 because it is simple. From the statement in part 1 for each node $j \in M_n^i$ there is a path $i \rightsquigarrow j$ in G_n^i with length at most $\mathbf{D_n^{i,j}}$. In the resulting tree after running Dijkstra, we can infer there is a path $i \rightsquigarrow j$ with length at most $\mathbf{D_n^{i,j}}$. Because there are $n-1$ nodes in M_n^i, the tree constructed has at least n nodes including node i. From property 1, it follows that the tree constructed is at least an n-hop minimum tree.

We now prove part 1. Order the nodes in M_n^i in non-decreasing order. The proof is by induction on the sequence of elements in M_n^i. The base case is true because for m_1, the first element of M^i, $l_{m_1}^i = min\{l_k^i | k \in N^i\}$ and $l_{m_1}^i = \mathbf{D_1^{i,m_1}}$. As induction hypothesis, let the statement hold for the first $m-1$ elements of M_n^i. Consider the m-th element $j \in M_n^i$. Let K be the highest priority neighbor for which $D_{jK}^i + l_K^i = min\{D_{jk}^i + l_k^i | k \in N^i\}$. At most $m-1$ nodes in T_K^i can have lesser or equal distance than j which implies path $K \rightsquigarrow j$ exists with at most $m-1$ hops. Let v be the neighbor of j in T_K^i. Then the path $K \rightsquigarrow v \rightarrow j$ has at most $m-1$ hops. Because T_K^i is at least a $(n-1)$-hop minimum tree, the link $v \rightarrow j$ must agree with $\mathbf{G}$. Since $D_{vK}^i + l_K^i < D_{jK}^i + l_K^i$, from induction hypothesis there is a path $i \rightsquigarrow v$ in G_n^i such that the length is at most $\mathbf{D_n^{i,v}}$.

Now we need to show that the preferred neighbor for v is also K, so that the link $v \rightarrow j$ will be included in the construction of G_n^i, thus ensuring the existence of the path $i \rightsquigarrow j$ in G_n^i. If some neighbor K' other than K is the preferred neighbor for v then one of the following two conditions should hold: (a) $D_{vK'}^i + l_{K'}^i < D_{vK}^i + l_K^i$ or (b) $D_{vK'}^i + l_{K'}^i = D_{vK}^i + l_K^i$ and priority of K' is greater than priority of K.

Case (a): Because $D_{jK}^i + l_K^i \leq D_{jK'}^i + l_{K'}^i$ it follows that the path $v \rightsquigarrow j$ in $T_{K'}^i$ is greater than cost of $v \rightarrow j$ in $\mathbf{G}$ which implies that $T_{K'}^i$ is not a $(n-1)$ hop minimum tree – a contradiction of assumption. Therefore $D_{vK}^i + l_K^i = min\{D_{vk}^i + l_k^i | k \in N^i\}$.

Case (b): Let Q_j be the set of neighbors that give the minimum distance for j, i.e., for each $k \in Q_j$, $D_{jk}^i + l_k^i = min\{D_{jk}^i + l_k^i | k \in N^i\}$. Similarly, let Q_v be such that for each $k \in Q_v$, $D_{vk}^i + l_k^i = min\{D_{vk}^i + l_k^i | k \in N^i\}$. If $k \in Q_v$ and $k \notin Q_j$, then it follows from same argument as in case (a) that $v \rightsquigarrow j$ in T_k^i is greater than cost of $v \rightarrow j$ in $\mathbf{G}$ implying T_k^i is not a $(n-1)$-hop minimum tree – a contradiction of assumption. Because K has the highest priority among all members of Q_j and $Q_v \subseteq Q_j$ and $k \in Q_v$, K also has the highest priority among all members of Q_v. Therefore $Q_v \subseteq Q_j$. Also, from the same argument it can be inferred that $K \in Q_v$. This proves that $v \rightarrow j$ will be included in the construction of G_n^i. Because $\mathbf{D_n^{i,v}} + \mathbf{d_{vj}} = \mathbf{D_n^{i,j}}$ in $\mathbf{G}$, where $\mathbf{d_{vj}}$ is the final cost of link $v \rightarrow j$, and length of $i \rightsquigarrow v$ in G_n^i is less than or equal to $\mathbf{D_n^{i,v}}$ from induction hypothesis, we have length of $i \rightsquigarrow j$ in G_n^i less than or equal to $\mathbf{D_n^{i,j}}$. This proves part 1 of the theorem.

Proof of Theorem 3:

The proof is by induction on t_n, the global time when for each node i, T^i is at least n-hop minimum tree. Because the longest loop-free path in the network has at most $N-1$ links where N is number of nodes in the network, t_{N-1} is the time when every node has the shortest path to every other node. We need to show that t_{N-1} is finite. The base case is t_1, the time when every node has 1-hop minimum distance and because the adjacent link changes are notified within finite time, $t_1 < \infty$. Let $t_n < \infty$ for some $n < N$. Given that the propagation delays are finite each node will have each of its neighbors n-hop minimum tree in finite time after t_n. From Theorem 2 we can see that the node will have at least the $(n+1)$-hop minimum tree in finite time after t_n. Therefore, $t_{n+1} < \infty$. From induction we can see that $t_{N-1} < \infty$.

Proof of Theorem 6:

The convergence of MPATH follows directly from the convergence of PATH because the update messages in MPATH are only delayed a finite time as allowed at line 4 in algorithm PATH. Therefore, the distances D_j^i in MPATH also converge to shortest distances. Because changes to D_j^i are always reported to the neighbors and are incorporated by the neighbors in their tables in finite time $D_{jk}^i = D_j^k$ for $k \in N^i$ after convergence. From line 3a in MPATH, we observe that when node i becomes passive $FD_j^i = D_j^i$ holds true. Because all nodes are passive at convergence it follows that $S_j^i = \{k | D_{jk}^i < FD_j^i, k \in N^i\} = \{k | D_j^k < D_j^i, k \in N^i\}$.